

Python: module cdms.cdmsNode

[cdms.cdmsNode](#)

[index](#)

CDMS node classes

Modules

[cdms.CDML](#)
[MA](#)

[Numeric](#)
[cdtime](#)

[re](#)
[string](#)

[sys](#)

Classes

[CdmsNode](#)
[AttrNode](#)
[AxisNode](#)
[DatasetNode](#)
[DocLinkNode](#)
[DomElemNode](#)
[DomainNode](#)
[LinearDataNode](#)
[RectGridNode](#)
[VariableNode](#)
[XLinkNode](#)
[cdms.error.CDMSError\(exceptions.Exception\)](#)
[NotMonotonicError](#)

```
class AttrNode(CdmsNode):
    # Attribute node - only used as a placeholder during parse and write
    #   Attr nodes are not placed on the tree
    #
    # Two ways to create an Attr object:
    # (1) attr = AttrNode(name,value)
    #       datatype = sometype # optionally, to override intrinsic type
    # (2) attr = AttrNode(name,None)
    #       attr.setValueFromString(somestring,sometype)
```

Methods defined here:

__init__(self, name, value=None)

getDatatype(self)

```

getLength(self)

getValueAsString(self)

mapToExternal(self)
    # Note: mapToExternal is not called at init time, must be called
    #       if needed

setContentFromString(self, content)
    # Set content
    # This may be called multiple times, so append

setValueFromString(self, valString, datatype)
    # Map a string of a given datatype to a value
    # Returns ValueError if the conversion fails

write(self, fd=None, tablevel=0, format=1)
    # Write to a file, with formatting.
    # tablevel is the start number of tabs

```

Methods inherited from [CdmsNode](#):

```

add(self, child)
    # Add a child node

children(self)
    # Return a list of child nodes

getChildAt(self, index)
    # Get the child node at index k

getChildCount(self)
    # Get the number of children

getContent(self)
    # Get content

getExternalAttr(self, name)
    # Get an external attribute

getExternalAttrAsAttr(self, name)
    # Get an external attribute, as an Attr instance

getExternalDict(self)
    # Get a dictionary of external attributes, of form (value, dat

getIndex(self, node)
    # Get the index of a node

getParent(self)
    # Get the parent node

```

```

isLeaf(self)
    # True iff node is a leaf node

removeChildAt(self, index)
    # Remove and return the child at index k

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
    # Set the external attribute dictionary. The input dictionary
    # is of the form {name:value,...} where value is a string.

validate(self, idtable=None)
    # Validate attributes

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: val"
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

write_raw(self, fd=None)
    # Write to a file without formatting.

```

```

class AxisNode(CdmsNode)
    # Coordinate axis

```

Methods defined here:

```

__init__(self, id, length, datatype='Long', data=None)
    # If datatype is None, assume values [0,1,...,length-1]
    # data is a Numeric array, if specified

__len__(self)

equal(self, axis)
    # Test if axis data vectors are equal

extend(self, axis, isreltime=0, allowgaps=0)
    # Extend axes. 'isreltime' is true iff
    # the axes are relative time axes
    # If allowgaps is true, allow gaps when extending linear vect

getContent(self)

```

```

    # Get the content string: the data values if the representation
    # is as a vector, or an empty string otherwise

getData(self)
    # Get the data as an array

isClose(self, axis, eps)
    # Test if axis data vectors are element-wise close
    # True iff for each respective element a and b, abs((b-a)/b) <

mapToExternal(self)
    # Map to external attributes

monotonicity(self)
    # Test for strict monotonicity.
    # Returns CdNotMonotonic, CdIncreasing, CdDecreasing, or CdSi

setContentFromString(self, datastring)
    # Set data from content string
    # The content of an axis is the data array.

setData(self, data)
    # Set the data as an array, check for monotonicity

setLinearData(self, linearNode, partition=None)
    # Set the data as a linear vector
    # If the partition is set, derive the vector length from it

setPartitionFromString(self, partstring)
    # Set the partition from a string. This does not
    # set the external string representation

```

Methods inherited from [CdmsNode](#):

```

add(self, child)
    # Add a child node

children(self)
    # Return a list of child nodes

getChildAt(self, index)
    # Get the child node at index k

getChildCount(self)
    # Get the number of children

getExternalAttr(self, name)
    # Get an external attribute

getExternalAttrAsAttr(self, name)
    # Get an external attribute, as an Attr instance

```

```

getExternalDict(self)
    # Get a dictionary of external attributes, of form (value,datatype)

getIndex(self, node)
    # Get the index of a node

getParent(self)
    # Get the parent node

isLeaf(self)
    # True iff node is a leaf node

removeChildAt(self, index)
    # Remove and return the child at index k

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
    # Set the external attribute dictionary. The input dictionary
    # is of the form {name:value,...} where value is a string.

validate(self, idtable=None)
    # Validate attributes

write(self, fd=None, tablevel=0, format=1)
    # Write to a file, with formatting.
    # tablevel is the start number of tabs

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: value"
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

write_raw(self, fd=None)
    # Write to a file without formatting.

```

class **CdnsNode**

Named node

Methods defined here:

__init__(self, tag, id=None, parent=None)

```
add(self, child)
    # Add a child node

children(self)
    # Return a list of child nodes

getChildAt(self, index)
    # Get the child node at index k

getChildCount(self)
    # Get the number of children

getContent(self)
    # Get content

getExternalAttr(self, name)
    # Get an external attribute

getExternalAttrAsAttr(self, name)
    # Get an external attribute, as an Attr instance

getExternalDict(self)
    # Get a dictionary of external attributes, of form (value, dat

getIndex(self, node)
    # Get the index of a node

getParent(self)
    # Get the parent node

isLeaf(self)
    # True iff node is a leaf node

mapToExternal(self)
    # Map to external attributes

removeChildAt(self, index)
    # Remove and return the child at index k

setContentFromString(self, content)
    # Set content from a string. The interpretation
    # of content is class-dependent

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
```

```

# Set the external attribute dictionary. The input dictionary
# is of the form {name:value,...} where value is a string.

validate(self, idtable=None)
    # Validate attributes

write(self, fd=None, tablelevel=0, format=1)
    # Write to a file, with formatting.
    # tablelevel is the start number of tabs

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: val"
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

write_raw(self, fd=None)
    # Write to a file without formatting.

```

```

class DatasetNode(CdmsNode)
    # Container object for other CDMS objects

```

Methods defined here:

```

__init__(self, id)

addId(self, id, child)
    # Add a child node with an ID

dump(self, path=None, format=1)
    # Dump to a CDML file.
    # path is the file to dump to, or None for standard output.
    # if format is true, write with tab, newline formatting

getChildNamed(self, id)
    # Get a child node from its ID

getIdDict(self)
    # Get the ID table

validate(self, idtable=None)
    # Validate the dataset and all child nodes

```

Methods inherited from CdmsNode:

```

add(self, child)
    # Add a child node

children(self)

```

```
# Return a list of child nodes

getChildAt(self, index)
    # Get the child node at index k

getChildCount(self)
    # Get the number of children

getContent(self)
    # Get content

getExternalAttr(self, name)
    # Get an external attribute

getExternalAttrAsAttr(self, name)
    # Get an external attribute, as an Attr instance

getExternalDict(self)
    # Get a dictionary of external attributes, of form (value, dat

getIndex(self, node)
    # Get the index of a node

getParent(self)
    # Get the parent node

isLeaf(self)
    # True iff node is a leaf node

mapToExternal(self)
    # Map to external attributes

removeChildAt(self, index)
    # Remove and return the child at index k

setContentFromString(self, content)
    # Set content from a string. The interpretation
    # of content is class-dependent

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
    # Set the external attribute dictionary. The input dictionary
    # is of the form {name:value,...} where value is a string.

write(self, fd=None, tablevel=0, format=1)
```

```

        # Write to a file, with formatting.
        # tablevel is the start number of tabs

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: val"
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

write_raw(self, fd=None)
    # Write to a file without formatting.

```

class ***DocLinkNode***(***CdmsNode***)

 # Link to a document

Methods defined here:

__init__(self, uri, content="")

mapToExternal(self)

 # Map to external attributes

Methods inherited from ***CdmsNode***:

add(self, child)

 # Add a child node

children(self)

 # Return a list of child nodes

getChildAt(self, index)

 # Get the child node at index k

getChildCount(self)

 # Get the number of children

getContent(self)

 # Get content

getExternalAttr(self, name)

 # Get an external attribute

getExternalAttrAsAttr(self, name)

 # Get an external attribute, as an Attr instance

getExternalDict(self)

 # Get a dictionary of external attributes, of form (value, dat

getIndex(self, node)

```

        # Get the index of a node

getParent(self)
    # Get the parent node

isLeaf(self)
    # True iff node is a leaf node

removeChildAt(self, index)
    # Remove and return the child at index k

setContentFromString(self, content)
    # Set content from a string. The interpretation
    # of content is class-dependent

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
    # Set the external attribute dictionary. The input dictionary
    # is of the form {name:value,...} where value is a string.

validate(self, idtable=None)
    # Validate attributes

write(self, fd=None, tablevel=0, format=1)
    # Write to a file, with formatting.
    # tablevel is the start number of tabs

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: val"
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

write_raw(self, fd=None)
    # Write to a file without formatting.

```

```
class DomElemNode(CdmsNode)
```

```
    # Domain element
```

Methods defined here:

```
__init__(self, name, start=None, length=None)
```

```
getName(self)
    # Get the name

mapToExternal(self)
    # Map to external attributes

setName(self, name)
    # Set the name

write(self, fd=None, tablevel=0, format=1)
    # Write to a file, with formatting.
    # tablevel is the start number of tabs
```

Methods inherited from [CdmsNode](#):

```
add(self, child)
    # Add a child node

children(self)
    # Return a list of child nodes

getChildAt(self, index)
    # Get the child node at index k

getChildCount(self)
    # Get the number of children

getContent(self)
    # Get content

getExternalAttr(self, name)
    # Get an external attribute

getExternalAttrAsAttr(self, name)
    # Get an external attribute, as an Attr instance

getExternalDict(self)
    # Get a dictionary of external attributes, of form (value, dat

getIndex(self, node)
    # Get the index of a node

getParent(self)
    # Get the parent node

isLeaf(self)
    # True iff node is a leaf node

removeChildAt(self, index)
    # Remove and return the child at index k

setContentFromString(self, content)
```

```

        # Set content from a string. The interpretation
        # of content is class-dependent

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
    # Set the external attribute dictionary. The input dictionary
    # is of the form {name:value,...} where value is a string.

validate(self, idtable=None)
    # Validate attributes

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: val"
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

write_raw(self, fd=None)
    # Write to a file without formatting.

```

class DomainNode(CdmsNode)

Domain

Methods defined here:

__init__(self)

Methods inherited from CdmsNode:

add(self, child)
 # Add a child node

children(self)
 # Return a list of child nodes

getChildAt(self, index)
 # Get the child node at index k

getChildCount(self)
 # Get the number of children

getContent(self)
 # Get content

```

getExternalAttr(self, name)
    # Get an external attribute

getExternalAttrAsAttr(self, name)
    # Get an external attribute, as an Attr instance

getExternalDict(self)
    # Get a dictionary of external attributes, of form (value, dat

getIndex(self, node)
    # Get the index of a node

getParent(self)
    # Get the parent node

isLeaf(self)
    # True iff node is a leaf node

mapToExternal(self)
    # Map to external attributes

removeChildAt(self, index)
    # Remove and return the child at index k

setContentFromString(self, content)
    # Set content from a string. The interpretation
    # of content is class-dependent

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
    # Set the external attribute dictionary. The input dictionary
    # is of the form {name:value,...} where value is a string.

validate(self, idtable=None)
    # Validate attributes

write(self, fd=None, tablevel=0, format=1)
    # Write to a file, with formatting.
    # tablevel is the start number of tabs

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: val
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

```

```
write_raw(self, fd=None)
    # Write to a file without formatting.
```

```
class LinearDataNode(CdmsNode)
    # Linear data element
```

Methods defined here:

```
__getitem__(self, index)
    # Get an indexed value

__init__(self, start, delta, length)

__len__(self)

concatenate(self, linearNode, allowgaps=0)
    # Concatenate linear arrays, preserving linearity
    # If allowgaps is set, don't require that the linear arrays b
    # Return a new linear node

equal(self, axis)
    # Equality of linear vectors

equalVector(self, ar)
    # Equality of linear vector and array

isClose(self, axis, eps)
    # Closeness of linear vectors

isCloseVector(self, ar, eps)
    # Closeness of linear vector and array

mapToExternal(self)
    # Map to external attributes

monotonicity(self)
    # Return monotonicity: CdNotMonotonic, CdIncreasing, CdDecreasing

toVector(self, datatype)
    # Return a vector representation, given a CDMS datatype
```

Data and other attributes defined here:

```
validDeltaTypes = [<type 'int'>, <type 'float'>, <type 'list'>]
```

```
validStartTypes = [<type 'int'>, <type 'float'>, <type 'comptime'>, <type 'reltime'>]
```

Methods inherited from CdmsNode:

```

add(self, child)
    # Add a child node

children(self)
    # Return a list of child nodes

getChildAt(self, index)
    # Get the child node at index k

getChildCount(self)
    # Get the number of children

getContent(self)
    # Get content

getExternalAttr(self, name)
    # Get an external attribute

getExternalAttrAsAttr(self, name)
    # Get an external attribute, as an Attr instance

getExternalDict(self)
    # Get a dictionary of external attributes, of form (value, dat

getIndex(self, node)
    # Get the index of a node

getParent(self)
    # Get the parent node

isLeaf(self)
    # True iff node is a leaf node

removeChildAt(self, index)
    # Remove and return the child at index k

setContentFromString(self, content)
    # Set content from a string. The interpretation
    # of content is class-dependent

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
    # Set the external attribute dictionary. The input dictionary
    # is of the form {name:value,...} where value is a string.

validate(self, idtable=None)

```

```

    # Validate attributes

write(self, fd=None, tablevel=0, format=1)
    # Write to a file, with formatting.
    # tablevel is the start number of tabs

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: val"
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

write_raw(self, fd=None)
    # Write to a file without formatting.

```

class *NotMonotonicError*(cdms.error.CDMSError)

Method resolution order:

[NotMonotonicError](#)
[cdms.error.CDMSError](#)
[exceptions.Exception](#)

Methods inherited from [cdms.error.CDMSError](#):

[__init__](#)(self, args='Unspecified error from package cdms')
[__str__](#)(self)

Methods inherited from [exceptions.Exception](#):

[__getitem__](#)(...)

class *RectGridNode*(CdmsNode)

Rectilinear lat-lon grid

Methods defined here:

[__init__](#)(self, id, latitude, longitude, gridtype='unknown', order='yx', mask=None)
 # Create a grid
 # All arguments are strings

mapToExternal(self)
 # Map to external attributes

Methods inherited from [CdmsNode](#):

add(self, child)

```

        # Add a child node

children(self)
    # Return a list of child nodes

getChildAt(self, index)
    # Get the child node at index k

getChildCount(self)
    # Get the number of children

getContent(self)
    # Get content

getExternalAttr(self, name)
    # Get an external attribute

getExternalAttrAsAttr(self, name)
    # Get an external attribute, as an Attr instance

getExternalDict(self)
    # Get a dictionary of external attributes, of form (value,dat

getIndex(self, node)
    # Get the index of a node

getParent(self)
    # Get the parent node

isLeaf(self)
    # True iff node is a leaf node

removeChildAt(self, index)
    # Remove and return the child at index k

setContentFromString(self, content)
    # Set content from a string. The interpretation
    # of content is class-dependent

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
    # Set the external attribute dictionary. The input dictionary
    # is of the form {name:value,...} where value is a string.

validate(self, idtable=None)
    # Validate attributes

```

```

write(self, fd=None, tablevel=0, format=1)
    # Write to a file, with formatting.
    # tablevel is the start number of tabs

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: val"
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

write_raw(self, fd=None)
    # Write to a file without formatting.

```

class **VariableNode**(CdmsNode)

```

# Spatio-temporal variable
# Two ways to create a variable:
# (1) var = VariableNode(id, datatype, domain)
# (2) var = VariableNode(id, datatype)
#     var.setDomain(domain)

```

Methods defined here:

```

__init__(self, id, datatype, domain)
    # Create a variable.
    # If validate is true, validate immediately

getDomain(self)
    # Get the domain

mapToExternal(self)
    # Map to external attributes

setDomain(self, domain)
    # Set the domain

```

Methods inherited from CdmsNode:

```

add(self, child)
    # Add a child node

children(self)
    # Return a list of child nodes

getChildAt(self, index)
    # Get the child node at index k

getChildCount(self)
    # Get the number of children

```

```

getContent(self)
    # Get content

getExternalAttr(self, name)
    # Get an external attribute

getExternalAttrAsAttr(self, name)
    # Get an external attribute, as an Attr instance

getExternalDict(self)
    # Get a dictionary of external attributes, of form (value, dat

getIndex(self, node)
    # Get the index of a node

getParent(self)
    # Get the parent node

isLeaf(self)
    # True iff node is a leaf node

removeChildAt(self, index)
    # Remove and return the child at index k

setContentFromString(self, content)
    # Set content from a string. The interpretation
    # of content is class-dependent

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
    # Set the external attribute dictionary. The input dictionary
    # is of the form {name:value,...} where value is a string.

validate(self, idtable=None)
    # Validate attributes

write(self, fd=None, tablevel=0, format=1)
    # Write to a file, with formatting.
    # tablevel is the start number of tabs

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: val
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

```

```
write_raw(self, fd=None)
    # Write to a file without formatting.
```

```
class XLinkNode(CdmsNode)
```

```
    # Link to an external element
```

Methods defined here:

```
__init__(self, id, uri, contentRole, content=")
```

```
mapToExternal(self)
```

```
    # Map to external attributes
```

```
Methods inherited from CdmsNode:
```

```
add(self, child)
```

```
    # Add a child node
```

```
children(self)
```

```
    # Return a list of child nodes
```

```
getChildAt(self, index)
```

```
    # Get the child node at index k
```

```
getChildCount(self)
```

```
    # Get the number of children
```

```
getContent(self)
```

```
    # Get content
```

```
getExternalAttr(self, name)
```

```
    # Get an external attribute
```

```
getExternalAttrAsAttr(self, name)
```

```
    # Get an external attribute, as an Attr instance
```

```
getExternalDict(self)
```

```
    # Get a dictionary of external attributes, of form (value, dat
```

```
getIndex(self, node)
```

```
    # Get the index of a node
```

```
getParent(self)
```

```
    # Get the parent node
```

```
isLeaf(self)
```

```
    # True iff node is a leaf node
```

```
removeChildAt(self, index)
```

```
    # Remove and return the child at index k
```

```

setContentFromString(self, content)
    # Set content from a string. The interpretation
    # of content is class-dependent

setExternalAttr(self, name, value, datatype=None)
    # Set an external attribute

setExternalAttrFromAttr(self, attr)
    # Set an external attribute
    # 'attr' is an Attr object

setExternalDict(self, dict)
    # Set the external attribute dictionary. The input dictionary
    # is of the form {name:value,...} where value is a string.

validate(self, idtable=None)
    # Validate attributes

write(self, fd=None, tablevel=0, format=1)
    # Write to a file, with formatting.
    # tablevel is the start number of tabs

write_ldif(self, parentdn, userAttrs=[], fd=None, format=1)
    # Write an LDIF (LDAP interchange format) entry
    # parentdn is the parent LDAP distinguished name
    # userAttrs is a string or list of strings of form "attr: val"
    # A trailing newline is added iff format==1
    # Note: unlike write, this does not write children as well

write_raw(self, fd=None)
    # Write to a file without formatting.

```

Functions

```

mapIllegalToEntity(matchobj)
    # Map illegal XML characters to entity references:
    # '<' --> &lt;;
    # '>' --> &gt;;
    # '&' --> &amp;;
    # '"' --> &quot;;
    # '\'' --> &apos;;
    # all other illegal characters are removed #"

```

Data

```

CdAny = 'Any'
CdArray = 'Array'
CdByte = 'Byte'

```

```
CdChar = 'Char'  
CdDatatypes = ['Char', 'Byte', 'Short', 'Int', 'Long', 'Float', 'Double', 'String']  
CdDecreasing = 1  
CdDouble = 'Double'  
CdFloat = 'Float'  
CdFromObject = 'FromObject'  
CdGridtypes = ['unknown', 'gaussian', 'uniform']  
CdIncreasing = -1  
CdInt = 'Int'  
CdLinear = 2  
CdLong = 'Long'  
CdNotMonotonic = 0  
CdScalar = 'Scalar'  
CdShort = 'Short'  
CdSingleton = 2  
CdString = 'String'  
CdToNumericType = {'Byte': 'l', 'Char': 'c', 'Double': 'd', 'Float': 'f', 'Int': 'i', 'Long': 'l', 'Short': 's'}  
CdVector = 1  
DuplicateIdError = 'Duplicate identifier: '  
GaussianGridType = 'gaussian'  
InvalidArgumentError = 'Invalid argument: '  
InvalidDatatype = 'Invalid datatype: '  
InvalidGridtype = 'Invalid grid type: '  
InvalidIdError = 'Invalid identifier: '  
NotMonotonic = 'Result array is not monotonic '  
NumericToCdType = {'l': 'Byte', 'c': 'Char', 'd': 'Double', 'f': 'Float', 'i': 'Int', 'l': 'Long', 's': 'Short'}  
StringTypes = (<type 'str'>, <type 'unicode'>)  
UniformGridType = 'uniform'  
UnknownGridType = 'unknown'
```